Internet Address Space

Peter J. Denning
3/9/19

The OS manages a large number of users and their digital objects (files, images, music, videos, memory regions, etc). Every digital object has a name -- a string of bits or symbols that designates it and allows users to access it. Names come in many forms. The most common are:

1. *Addresses* -- bit strings that designate locations in storage media and are processed by the hardware.

2. *Handles* (pointers) -- strings that point to names. A handle points to a name, not a location.

3. *Symbolic strings* -- strings of ASCII symbols chosen by users to name their objects.

4. *Pathnames* -- sequences of symbolic strings tracing a path through the *directory* structure (for example, `/usr/dev/printer` in UNIX or LINUX).

5. *Descriptors* – strings of the form (base, length) that designate a memory region.

6. *Internet host names* -- strings of the form *host.domain* that identify a unique host within the given domain (for example, `nps.edu`).

7. *Internet URLs* -- strings of the form hostname/pathname that identify a file on the host by giving its pathname in the host's directory structure (for example, `jay.nps.edu/usr/dev/printer`).

8. *Capabilities* -- unforgeable certificates that name an object and grant a particular access to the object. Objects stored in directory systems have access control lists (ACL) specifying which users have access and what each of their permissions is. Permissions must be checked on each requested access; verifying them with an ACL is expensive because of searching the ACL. A capability can be derived from an ACL and is like a ticket granting immediate access. Processing a capability is very fast because the ACL check has already been done and is embedded in the capability.

We have already encountered the first five kinds of addresses in the microkernel. In this module, we examine two schemes for naming objects in a distributed network such as the Internet: the URL and the capability. In all cases, the OS is engaged in mapping names to the locations of objects.

The term *binding* means that the OS has recorded an association of name-to-object in its data structures. Static binding means that the association does not change during execution. Dynamic binding means that the OS can change the association during execution. Dynamic binding allows great flexibility in moving things around without changing their names -- thereby achieving some degree of *location independence* -- and is used throughout the OS.

Location independence is important when a user is part of a large network. The user wants access to files and services from anywhere in the network, with names that do not change if the file, service, or user's location moves. Cloud computing relies heavily on this principle. For example, the binding of an Internet hostname to a physical host is recorded in the Domain Name Service (DNS); the host owner can move the host to a new location with a new Internet Protocol (IP) address and simply update the registration in the DNS. The binding of a pathname to a physical file is composed of two smaller bindings: the pathname points to an entry in a directory, which contains a handle pointing to the actual file. The two-level binding enables sharing the file because many users can have a copy of the same handle in their directories, but with different pathnames.

Access Control

All digital objects are subject to access controls specified by their owners. Access controls say who is allowed what kinds of access to a specific object. For example, to allow user rlb to read and write a file named "temp", an owner would place the entry "rlb: rw" in the access control list (ACL) of file "temp". ACLs were first used in the file systems of the 1960s. They are convenient and any updates are immediately effective. But they are also not very efficient because they have to be stored in separate files; opening a file thus means to open two files (the file and its ACL).

UNIX tried to eliminate the separate ACL by collapsing all file permissions into 9 bits. UNIX divides users into three kinds: self, group, and world. "Self" is the owner, "group" is a list of users specified by the owner, and "world" is everyone else. Three bits *rwx* for read-write-execute permission are specified for each group. These 9 bits are stored in the directory entry for the file, where they can be rapidly checked by the file system when a process wants to access the file.

In 1966 Jack Dennis and Earl Van Horn of MIT invented another way to manage access: capabilities. A *capability* is an unforgeable pointer that contains the allowed access codes to an object. Let's say process *P* creates a new file *F* with open, read, and write permissions. The OS makes a capability that says *(file, orw, F)* and places it in a data structure attached to the process, called C-list (for capability list). The C-list is in kernel space and not accessible to the process. The capability is tagged with the type of object it points to, so that it will only work with object manager of that type. Let's say the above capability is at position *i* in the C-list of process *P*; *P* can open the file with the kernel call `OPEN_FILE(`*i*`)`. The `OPEN_FILE` program gets the capability (*file, orw, F*) from position *i* in *P*'s C-list, validates that its tag is "file" and that the open permission bit is on, and then opens the file. Thus, you can think of capabilities as pre-approved accesses could be in an ACL but are now encoded into a capability for rapid processing.

Internet Names

The designers of the Internet created a hierarchical system for naming all the hosts (individual computers) in the network. The topmost domains were for industry sectors such as .com, .edu, and .org. Within each sector is a registrar that hands out host names within the sector; for example, the .edu registrar gave the Naval Postgraduate School the name nps.edu. In its turn the internal NPS registrar can designate internal subdomains such as intranet.nps.edu. Registrars are not allowed to duplicate names. This scheme gives each host a unique name and spreads the workload of maintaining names across many registrars.

The designers also created a system of numerical addresses for hosts, known as IP addresses. They are expressed as four blocks of three digits, such as 172.20.108.202. These addresses are used by the Internet Protocol (IP) to rapidly route messages to hosts. The Internet domain name service (DNS) is a distributed database that maps host names (such as python.nps.edu) to IP addresses.

The designers of the Internet also invented the URL (uniform resource locator), which is a string of the form hostname/pathname. The hostname is the name of a host as described above. The pathname is the path of a digital object in the host's directory system. The URL gives a unique name for every object in the Internet. To read the file hostname/pathname, the user employs a protocol http by typing http://hostname/pathname to a browser; the protocol sends messages to the target host, which looks up the file in its local system, checks that read access is permitted by the access code, and then transmits the contents of the file back to the calling browser.

Capabilities

A capability contains an object type tag, an access code, and a handle (unique pointer) to an object. The tag tells what type of object is designated by the capability – for example, file or process. Thus, a capability c = (file, orw, handle) says that the user holding it has open, read, and write access to the file designated by the handle pointer. A capability c = (process, suspend, handle) says that the user holding it has the authority to suspend the process designated by the handle.

As discussed above, a capability is an unforgeable pointer to an object that contains a validated access code for the capability's owner. Unforgeable means that no one can alter the capability. Capabilities are stored in a process's C-list which is in protected kernel space, guaranteeing their unforgeability.

Capabilities can be shared across the network. To guarantee that a capability remains unforgeable when **outside** the protection of its operating system, we add an identifier of the object's home machine and a cryptographic checksum

(tag, access, host, handle, check)

where

tag indicates the type of object (such as file),

access the permitted operations (such as read),

host the unique identifier of a host (e.g., its IP address)

handle is a unique-for-all-time name (such as a timestamp combined with MAC address) composed at the time the capability was created , and

check is a cryptographic checksum to verify that the capability has not been modified since its creation.

With this encoding, a process wanting access to a file calls `OPEN_FILE` its local file system, which then makes remote procedure call to `OPEN_FILE` on the file's host machine. The host can verify that the capability it receives in from the remote procedure call has not been altered since its creation.

The C-list is a powerful tool for confinement. Confinement means that an untrusted process can be confined to a minimal memory space and set of objects to do its advertised job. If the untrusted process tries to use more memory than it was allocated or access objects not in its C-list, the OS will stop it before it can do damage. Modern operating systems approximate the C-list method of confinement with sandboxes, which restrict application programs to limited memory and an authorized set of objects.

In the reference model of os99.pdf, all objects above the IPC level are named by capabilities. This allows for fast processing and for sharing objects by sharing capabilities. Capabilities are a model for how the OS keeps track of all the objects in the system and allows users to invoke only authorized actions on those objects.